Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Mitja Jeseničnik Kotnik

Razvoj učinkovitih lokacijsko zavednih mobilnih aplikacij na platformah Android in iOS

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Analizirajte tehnične možnosti za uporabo lokacije pri razvoju mobilnih aplikacij, identificirajte različne pristope in algoritme ter jih opišite. Identificirajte in primerjate različne načine implementacije zavedanja lokacije v mobilnih aplikacijah. Proučite načine vključevanja lokacije v operacijskih sistemih Android in iOS. Izdelajte primer lokacijsko zavedne mobilne aplikacije na obeh prej omenjenih operacijskih sistemih in analizirajte ter primerjajte učinkovitost delovanja.

Zahvaljujem se mentorju prof. dr. Matjažu Branku Juriču za pomoč in usmerjanje med izdelavo diplomskega dela in asist. Melaniji Vezočnik za natančen pregled diplomskega dela. Družini se zahvaljujem za podporo. Posebna zahvala gre partnerki Živi, ki me je podpirala in verjela vame, ter hčeri Juliji, ki je bila dodaten navdih za zaključek diplomskega študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
GPS	global positioning system	globalni sistem pozicioniranja
A-GPS	assisted global positioning system	podprti globalni sistem pozicioniranja
WPS	Wi-Fi positioning system	Wi-Fi pozicijski sistem
MAC	media access control	nadzor dostopa medija
RSSI	received signal strength indication	indikator moči prejetega signala
UUID	universally unique identifier	univerzalno enolični identifikator
BLE	bluetooth low energy	nizka energija bluetooth
CRM	customer relationship management	upravljanje odnosov s strankami
API	application programming interface	aplikacijski programski vmesnik
IDE	integrated development environment	integrirano razvojno okolje
APK	Android package kit	Android komplet pakiranja
IP	internet protocol	internetni protokol

Povzetek

Naslov: Razvoj učinkovitih lokacijsko zavednih mobilnih aplikacij na platformah Android in iOS

Avtor: Mitja Jeseničnik Kotnik

Poznavanje lokacije mobilnih naprav postaja v zadnjih letih zaradi vse večjega števila potrošniško usmerjenih aplikacij čedalje pomembnejše. Glede na lokacijo nam omogoča personalizirati oglase za uporabnika, prikazati vremenske napovedi in najti vse bližnje ustanove, ki jih v danem trenutku potrebujemo. V diplomskem delu smo predstavili različne načine implementacije lokacijske zavednosti v mobilnih aplikacijah na platformah Android in iOS. Implementirali smo aplikacijo, v kateri smo lokacijsko zavednost omogočili s pomočjo lokacijskih področij in jo preizkusili na praktičnem primeru. Med nastajanjem tega dela smo ugotovili, da je natančnost odvisna od hitrosti gibanja naprave in radija lokacijskega področja.

Ključne besede: mobilne naprave, ios, android, lokacija, GPS, lokacijska področja.

Abstract

Title: Developing efficient location-aware mobile applications on Android and iOS platforms

Author: Mitja Jeseničnik Kotnik

Knowing the location of mobile devices is becoming more and more important in recent years due to an increasing number of consumer-oriented applications. Depending on the location, it allows us to personalize ads for the user, display weather forecasts, and find all nearby institutions that we need at a given moment. In this thesis, we presented various ways to implement location awareness in mobile applications on Android and iOS platforms. We implemented an application in which the location awareness was made possible through geofences. In the course of this work, we established that the accuracy depends on the speed of movement of the device and the radius of the location region.

Keywords: mobile devices, ios, android, location, GPS, geofence.

Poglavje 1

Uvod

1.1 Motivacija

Današnje aplikacije uporabljajo različne senzorje, ki so jim na voljo, in zajemajo senzorje za zaznavanje premikanja, okolice in pozicije naprave, kot so med drugim žiroskop, pospeškometer, magnetometer. Med njimi je lokacijski senzor GPS, ki se danes uporablja v večini aplikacij. Ta se lahko uporablja za zaznavanje položaja uporabnika aplikacije na prostem. Za uporabo v notranjih prostorih uporabljamo določanje lokacije s pomočjo svetilnikov, saj se GPS v notranjih prostorih ne uporablja zaradi sten in streh, ki lahko signal oslabijo in razpršijo. Primer uporabe za zaznavanje lokacije v notranjih prostorih so npr. trgovine, kjer s pomočjo aplikacije vodijo uporabnika po nakupih. Primer uporabe na prostem pa je primer, ko s pomočjo lokacij pošiljamo uporabniku obvestila o ponudbah v trgovini, ko se ta nahaja v bližini.

1.2 Cilji

Cilj diplomskega dela je raziskati vse tehnologije, ki nam omogočajo pridobivanje lokacije na mobilnih napravah. Naslednji cilj je, da raziščemo in primerjamo dva pristopa implementacije lokacijske zavednosti v mobilnih

aplikacijah. Lokacijsko zavednost implementiramo s pomočjo lokacijskih področij, ki jo nato testiramo v realnem okolju. Z implementirano aplikacijo testiramo delovanje v različnih pogojih in ocenimo delovanje.

1.3 Struktura diplome

Struktura dela je sledeča. V poglavju 2 raziščemo različne tehnologije, ki omogočajo pridobivanje lokacije mobilne naprave. Te tehnologije natančno opišemo in predstavimo njihovo delovanje ter uporabnost. Poglavje 3 predstavi različne možnosti za implementacijo lokacijske zavednosti v mobilne aplikacije. Predstavimo delovanje lokacijske zavednosti s pomočjo svetilnikov, ki delujejo na bluetooth tehnologiji, in delovanje lokacijskih področij. V poglavju 4 opišemo implementacijo mobilne aplikacije, ki deluje s pomočjo lokacijskih področij na Android in iOS operacijskem sistemu. Primerjamo implementaciji med operacijskima sistemoma in raziščemo, kako te razlike vplivajo na razvoj mobilne aplikacije. Poglavje 5 temelji na testiranju aplikacije, ki jo razvijemo za različna operacijska sistema.

Poglavje 2

Delovanje lokacijske zavednosti v mobilnih napravah

Lokacijsko zavednost nam omogoča naprava, katere komponente lahko aplikacija uporabi, da določi, kje se uporabnik nahaja. Ta pojem se danes največ uporablja v informacijski tehnologiji, kjer lahko takšno delovanje implementiramo na mobilne naprave, prav tako pa tudi v spletne aplikacije, kjer lahko potem na podlagi tega dostavljamo uporabniku lokacijsko targetirane informacije [33].

Različne tehnologije implementirajo tudi različne načine pridobivanja informacij o lokaciji. Pogostejši načini pridobivanja informacije o lokaciji so GPS, WiFi ter mobilna omrežja. Aplikacijo, ki na podlagi senzorskih podatkov naprave določi lokacijo, imenujemo lokacijsko zavedna aplikacija.

Aplikacije, ki uporabljajo lokacijsko zavednost, so uporabne za različna področja. Najpogosteje se uporabljajo za dostavljanje ciljnih oglasov. Druga področja uporabe se pogosto pojavljajo tudi v turizmu in socialnih omrežjih. Vsaka uporaba lokacijske zavednosti v aplikacijah je specifična in za vsako izmed njih lahko izbiramo med različnimi možnostmi implementacije. Od implementacije s pomočjo WiFi do uporabe GPS. Zaradi tega moramo pri razvoju lokacijsko zavedne aplikacije že v naprej predvideti, kako se bo ta uporabljala in na podlagi tega izbrati ustreznejši način pridobivanja loka-

cijskih podatkov. Sedaj bomo predstavili štiri najpogostejše tehnologije za pridobivanje podatkov o lokaciji [32]. Predstavlimo tehnologijo za pridobivanje lokacije z uporabo GPS, mobilnega omrežja, WiFi in Bluetooth. V poglavju predstavimo tudi najpogostejše pristope in algoritme, ki so nam na voljo za izračun razdalje mobilne naprave od določene točke, kjer se nahaja oddajnik.

2.1 Načini določanja lokacije glede na uporabo tehnologij

Glede na uporabo tehnologij lahko za določanje lokacije uporabimo številne tehnologije. Najpogostejše so GPS, mobilna omrežja, Wifi in Bluetooth, ki jih opišemo v nadaljevanju. Predstavimo delovanje tehnologije GPS, ki se danes uporablja pretežno za namene določanja lokacije sprejemnikov GPS signala. Nato podrobneje spoznamo, kako deluje pridobivanje lokacije z uporabo mobilnega omrežja. Sledi predstavitev delovanja pridobivanja lokacije z uporabo tehnologije WiFi.

2.1.1 Pridobivanje lokacije z uporabo GPS

GPS na telefonih deluje s pomočjo radijskih valov. Uporablja satelite, ki krožijo okoli zemlje v orbiti, in komunicira z njimi. S pomočjo prejetih radijskih valov lahko določimo svoj položaj tako, da naprava sprejme signal od najmanj treh satelitov iz orbite in na podlagi tega izračuna svoj položaj s pomočjo trilateracije, ki jo prikazuje slika 2.3. Vsak GPS satelit pošilja proti Zemlji sporočilo, ki vsebuje natančen časovni žig in svojo lokacijo v času oddaje sporočila. Ko naprava sprejme to sporočilo, lahko na podlagi časovnega žiga, kdaj je bilo sporočilo poslano in kdaj prejeto, izračuna razdaljo med napravo in satelitom [28]. Ker sateliti zraven pošiljajo tudi podatke o svoji trenutni lokaciji, lahko s pomočjo trilateracije izračunamo svojo trenutno lokacijo. Okoli vsakega izmed treh satelitov lahko izrišemo kroglo, ki ima

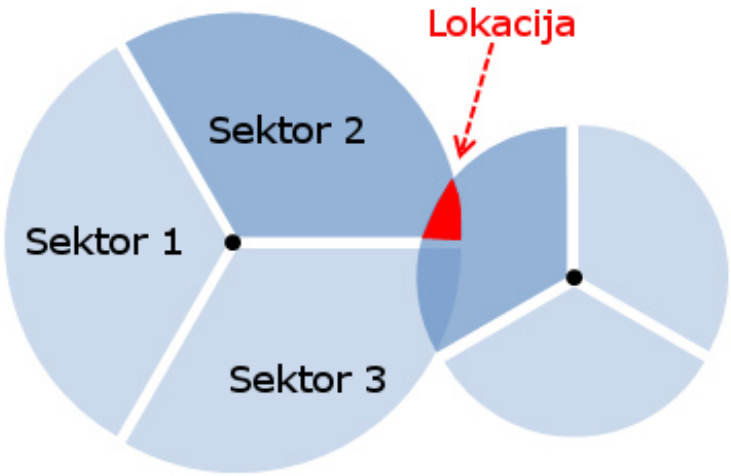
radij razdalje do naprave. Na presečišču teh treh krogel dobimo točko, ki predstavlja položaj naprave [47].

S pomočjo tega izračuna lahko določimo lokacijo naprave od 5 do 10 metrov natančno. Na hitrost pridobivanja lokacije pa lahko vplivajo tudi nekateri zunanji faktorji, kot na primer visoke stavbe in drevesa. Posledično lahko signal ob prehodu skozi stene in drevesa oslabi ali se odbija. Zaradi tega lahko traja pridobivanje GPS signala od 30 sekund do nekaj minut. V teh primerih nam boljše delovanje omogoča assisted GPS (A-GPS), ki s pomočjo najbližjih celičnih stolpov hitreje pridobi GPS signal. Deluje na način, da pridobi podatke o lokaciji satelitov od najbližjega celičnega stolpa. Stolpi imajo GPS sprejemnik, ki ves čas sprejemajo GPS signal in izračunavajo podatke o lokaciji satelitov. Pridobljeni podatki so potem posredovani mobilni napravi, ki lahko hitreje vzpostavi povezavo z GPS sateliti [45].

2.1.2 Pridobivanje lokacije z uporabo mobilnega omrežja

Lokacija se hitreje pridobi s pomočjo mobilnega omrežja kot s pomočjo GPS, ker uporablja celične stolpe, ki so na statičnih lokacijah. Deluje na podoben način kot GPS, torej s pomočjo trilateracije, ampak je njegova natančnost slabša v območjih s šibkim mobilnim omrežjem. Prav tako kot pri GPS je v primeru, da se uporablja samo en stolp, naša lokacija nezanesljiva, kadar želimo natančno lokacijo mobilne naprave. Stolpi imajo v večini primerov tri usmerjene antene, ki so postavljene v trikotnik. Torej lahko celični stolp pove, na katero anteno prihaja signal od mobilne naprave. Vsaka izmed anten pokriva 120° sektor, kjer je v središču stolp. S pomočjo tega okvirno določimo, kje se mobilna naprava nahaja, saj poznamo njeno oddaljenost od stolpa in sektor. Razdaljo naprave določimo glede na relativno moč signala naprave, torej dobimo radij, v katerem se nahaja naprava, in sektor zaradi treh anten, ki sprejemajo signale [37]. V primeru, da je v okolici večje število celičnih stolpov, lahko napravi določimo natančnejšo lokacijo. Za natančnejšo določitev lokacije naprave so potrebni trije celični stolpi, tako kot pri GPS, in nato še predstavitev pridobivanja lokacije mobilne naprave s

pomočjo Bluetooth tehnologije.



Slika 2.1: Izračun lokacije naprave s celičnimi stolpi s trilateracijo

V našem primeru, razvidnem iz slike 2.1, telefon sočasno pošilja signal na dva različna stolpa. V tem primeru je lokacija natančnejša kot v primeru, kjer je bil na voljo samo en stolp. Ker dva stolpa prejemata signal iste naprave, vemo, da se nahaja v eni izmed dveh točk, kjer se kroga sekata. Ker poznamo tudi sektor, kje se naprava nahaja, vemo, da je v točki, kjer se krog drugega stolpa seka s sektorjem 2 iz prvega stolpa. Posledično vidimo, da je lahko lokacija naprave s pomočjo sektorjev natančna tudi v primerih, kadar imamo na voljo samo dva stolpa.

2.1.3 Pridobivanje lokacije z uporabo WiFi

Pridobivanje lokacijskih informacij s strani WiFi je lahko tudi natančnejša kot pridobivanje s strani GPS. Velikokrat se zgodi, da nas aplikacije vprašajo, ali želimo spremeniti pridobivanje podatkov na WiFi za natančnejšo lokacijo. To se pogosto zgodi v bolj urbanih okoljih, kjer je veliko WiFi omrežij, ki oddajajo signal. Tukaj lahko pridobivanje GPS podatkov tudi ni mogoče, če se nahajamo v kakšnih trgovskih centrih ali na podzemni postaji.

Mobilne naprave, ki imajo WiFi in GPS, lahko pošiljajo informacije o WiFi omrežjih, ki se nahajajo v bližini. Te informacije zbira podjetje in jih hrani v svojih bazah. Eni takšnih sta Google[1] in Skyhook[2].

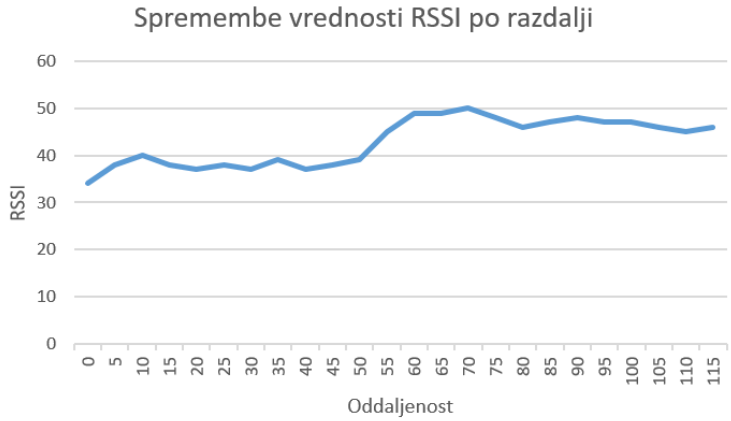
Deluje na način, da pošilja MAC naslov omrežne naprave, poleg tega pa tudi lokacijo, ki jo določi s pomočjo GPS. MAC naslov je identifikator omrežne naprave, ki je vpisana v strojno opremo naprave. Vsaka naprava ima unikaten MAC naslov, ki je sestavljen iz zaporedja črk in številk. Prvi štirje bajti naslova predstavljajo izdelovalca opreme in so določeni v naprej, preostale štiri bajte določi podjetje, ki proizvaja napravo. Pošiljanje informacij se zgodi vsakič, ko se naprava uporabi za določanje svoje lokacije s pomočjo GPS tako, da zraven še skenira vse bližnje javno dostopne WiFi točke. Ko pridobi vse te podatke, jih shrani v splet. Naslednjič, ko se kakšna mobilna naprava nahaja v bližini shranjenih WiFi naprav in nima zadovoljivega GPS signala, se lahko uporabi pridobivanje lokacije s pomočjo WiFi naprav [49].

Informacije o lokaciji WiFi naprav se neprestano osvežujejo s strani ponudnikov, kot so Google ali Apple[3], ter tako ponujajo natančnejše podatke o lokacijah svojim uporabnikom. Vse informacije o brezžičnih napravah, ki jih ti ponudniki zbirajo, so javno dostopne (MAC naslov naprave) in ne potrebujejo gesla, da bi ta način deloval.

Pridobivanje lokacije deluje tudi pri WiFi s trilateracijo, kot pri GPS in celičnih stolpih. S pomočjo WiFi pa je lahko lokacija natančnejša, saj so naprave manjše in imajo krajši domet kot celični stolpi [36].

2.1.4 Pridobivanje lokacije z uporabo Bluetooth

Pridobivanje lokacije preko bluetooth je uporabno takrat, ko GPS ne deluje. To je v primerih, ko smo v zaprtih prostorih, ali pa kadar varčujemo z baterijo. Pridobivanje lokacije s pomočjo bluetooth poteka tako, da imamo svetilnike na različnih mestih v prostoru, kjer oddajajo signal, da ga lahko mobilne naprave zaznajo. Te bluetooth naprave ne oddajajo kakšne fizične lokacije kot npr. GPS, ampak mobilna naprava od nje sprejme signal z indi-



Slika 2.2: Relacija med RSSI in razdaljo

katorjem moči prejetega signala (RSSI - received signal strength indication). RSSI nam pove moč signala, ki ga sprejme naprava v procentih. Višji kot je procent, močnejši je signal bluetooth naprave [38]. Če pred tem poznamo lokacijo bluetooth naprave, lahko s pomočjo te informacije glede na RSSI ugotovimo, kakšna je lokacija mobilne naprave. Podatek o lokaciji bluetooth naprave mora biti shranjen v dostopni podatkovni bazi ali na mobilni napravi. Ena glavnih težav pri RSSI je, da ne moremo določiti smeri, kje se nahaja bluetooth naprava glede na mobilno. V primeru gibanja in na podlagi RSSI trenda lahko določimo, ali postaja signal močnejši ali šibkejši [29], kar je razvidno iz grafa na sliki 2.2. Kadar postaja signal močnejši, pomeni, da se približujemo bluetooth napravi in obratno.

Ko imamo v prostoru več bluetooth naprav, lahko prav tako s pomočjo trilateracije izračunamo pozicijo mobilne naprave.

Bluetooth z nizko porabo energije

Bluetooth z nizko porabo energije (Bluetooth Low Energy) se uporablja za komunikacijo med mobilnimi napravami in svetilniki. To je brezžična tehnologija, ki se uporablja za prenos podatkov na kratkih razdaljah. Deluje na enak način kot njen predhodnik klasičen bluetooth, ampak z nižjo porabo

energije. Deluje lahko tudi do 3 leta z energijo ene gumbne baterije. Naprave, ki podpirajo BLE, delujejo na način, da pošiljajo periodično majhne pakete podatkov preko radijskih valov. Ta komunikacija je enosmerna in jo lahko zaznajo naprave, kot so pametni telefoni, ki podpirajo BLE in lahko preko tega sprožijo določene dogodke glede na svetilnik, ki oddaja. To so lahko različna potisna sporočila ali opozorila. Na vsakem svetilniku lahko nastavimo čas pošiljanja sporočil v milisekundah.

2.1.5 Natančnost različnih implementacij

Vsaka izmed predstavljenih tehnologij ima svoje prednosti in slabosti. S pomočjo primera uporabe lokacijske zavednosti v aplikaciji se bomo šele lahko odločili, kakšen način pridobivanja podatkov o lokaciji bomo uporabili. Najpomembnejša faktorja pridobivanja podatkov o lokaciji sta seveda hitrost in natančnost lokacije. Slednja je v tabeli 2.1 predstavljena za različne tehnologije [14].

`tehnologija`	zaprti prostori	odprti prostori	`natančnost`
`GPS`	ne	da	20–30m
`mobilno omrežje`	da	da	50–1000m
`WiFi`	da	da	3–10m/20–50m
`Bluetooth`	da	ne	3–10m

Tabela 2.1: Tabela primerov uporabe in natančnost lokacije naprave

2.2 Pristopi

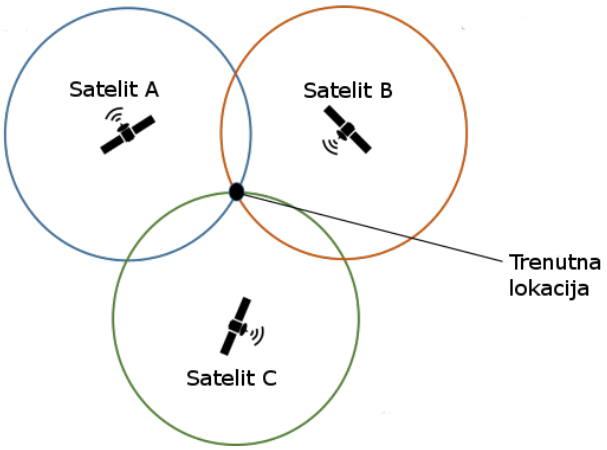
Za določanje lokacije mobilne naprave imamo na voljo različne pristope, ki jih v sledečem poglavju tudi predstavimo. Najprej spoznamo pristop trilateracije, ki mu sledi predstavitev pristopov neposredne bližine in lokacije na podlagi odtisov, ki, uporabljajo za določanje lokacije naprave, ki odda-

jajo signal. V naslednjem poglavju predstavimo pristop določanja lokacije s pomočjo zlivanja. V zaključku poglavja spoznamo računsko navigacijo.

2.2.1 Trilateracija

Za izračun lokacije s pomočjo trilateracije potrebujemo tri različne točke, ki v primeru iskanja lokacije predstavljajo satelite, WiFi dostopne točke ali svetilnike. Lateracija deluje tudi z manjšim številom točk, ampak je s tem njegova natančnost manjša. Trilateracija deluje tako, da izračuna oddaljenost od treh točk. V primeru na sliki 2.3 predstavimo izračun trenutne lokacije mobilne naprave glede na satelite. Kadar poznamo oddaljenost od satelita, izrišemo krog okoli satelita. V primeru trilateracije izrišemo tri kroge, katerih radij predstavlja oddaljenost mobilne naprave od satelita. Na točki, kjer se krogi sekajo, dobimo lokacijo mobilne naprave.

Pri izračunu s pomočjo trilateracije je njena slabost, da traja nekaj časa, da pridobi lokacijo naprave, saj mora zaznati signal treh različnih oddajnih točk. Medtem ko je njena prednost na splošno boljša natančnost določanja lokacije v primerjavi s pristopom neposredne bližine, ki ga opišemo v naslednjem razdelku.



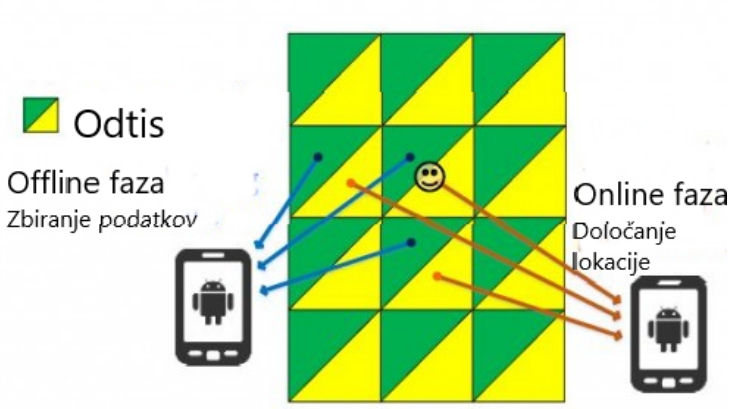
Slika 2.3: Izračun lokacije naprave s pomočjo trilateracije

2.2.2 Pristop neposredne bližine

Za pristop neposredne bližine uporabljamo naprave, kot so svetilniki, ki bazirajo na bluetooth tehnologiji. Pristop deluje po principu, da se ob bližanju določenemi napravi, ki oddaja signal, to napravo zazna in na podlagi tega določi trenutna lokacija uporabnika. Slabost tega pristopa je povečan znesek implementacije lokacijske zavednosti, saj je potrebno poleg implementacije aplikacije tudi kupiti svetilnik. Prednost pristopa pa je enostavno določanje lokacije v določenem območju, kjer imamo postavljene svetilnike.

2.2.3 Lokalizacija na podlagi odtisov

Lokalizacija na podlagi odtisov (*fingerprinting*) temelji na moči signala iz različnih dostopnih točk v dosegu. Za pravilno delovanje moramo imeti na voljo podatkovno bazo, v kateri hranimo podatke o napravi, kot je identifikacijska številka ali serijska številka in njihovi lokaciji. To pridobimo v *offline* fazi, kjer zbiramo podatke ter jih nato hranimo v podatkovno bazo. S pomočjo podatkovne baze in močjo signala ter identifikacijo dostopne točke lahko določimo lokacijo mobilne naprave. Tej fazi pravimo *online* faza [48].



Slika 2.4: Izračun lokacije naprave na podlagi odtisov

2.2.4 Zlivanje

Pristop zlivanja nam omogoča, da podatke pridobljene iz različnih senzorjev za lokacijo združimo [43]. Združeni podatki nam omogočijo natančnejšo določanje lokacije mobilne naprave. Združevanje podatkov poteka iz različnih senzorjev, kot so GPS, pospeškometer in magnetometer. Slabost tega pristopa je komunikacija med različnimi senzorji, kar zahteva večjo pasovno širino in obdelavo ter razumevanje večje količine podatkov.

2.2.5 Računska navigacija

Računska navigacija (*Pedestrian dead reckoning*) uporablja senzorje za zaznavo gibanja, kot sta med drugim pospeškometer in magnetometer. Z informacijami, ki jih prejme od teh senzorjev, lahko določimo smer gibanja mobilne naprave in razdaljo. Vsakič, ko se uporabnik z mobilno napravo premakne, se na podlagi meritev senzorjev določi razdalja, ki jo je prehodil, in s pomočjo magnetometra ali žiroskopa smer, kamor je šel. Glede na to lahko nato določimo, koliko in kam se je uporabnik mobilne naprave glede na prej znano lokacijo premaknil.

2.3 Algoritmi

Za izračun lokacije mobilne naprave imamo na voljo različne algoritme. V sledečem poglavju predstavimo algoritme, ki se v današnjih napravah pogosto uporabljajo za določanje lokacije. Najprej predstavimo algoritem čas prihoda, ki mu sledi algoritem časovna razlika prihoda. Oba delujeta na podlagi spremenljiv hitrosti signala in časa potovanja. Nato predstavimo še algoritem moč prejetega signala, ki pa za določanje lokacije, kot je razvidno iz imena uporablja moč prejetega signala.

2.3.1 Čas prihoda

Algoritem čas prihoda uporablja za določanje lokacije podatke o času, kdaj je bil signal s satelita poslan, čas, ko je prispel do naprave, in hitrost, kako hitro signal potuje [42]. Na podlagi teh podatkov lahko izračunamo razdaljo med satelitom in mobilno napravo d kot:

$$d = c \cdot (t_{prihod} - t_{poslan}), \tag{2.1}$$

kjer c označuje hitrost potovanja signala, t_{prihod} čas prihoda signala na mobilno napravo in t_{poslan} čas oddaje signala iz satelita.

Slabost tega algoritma je napaka v sinhronizaciji časa, saj v tem primeru pride do napačnega izračuna razdalje med satelitom in mobilno napravo. Medtem ko je prednost natančnost določanja lokacije.

2.3.2 Časovna razlika prihoda

Časovna razlika prihoda ne potrebuje časa, kdaj je bil signal poslan, temveč samo čas, kdaj je bil signal sprejet, in hitrost, s katero potuje [42]. Lokacijo določimo tako, da pošljemo signal iz mobilne naprave in ga različni sprejemniki prejmejo. Ko sprejemniki signal prejmejo, se na podlagi različnega časa prihoda izračuna razlika v razdalji med mobilno napravo in oddajniki $\Delta\ d$ kot:

$$\Delta d = c \cdot (\Delta t), \tag{2.2}$$

kjer c označuje hitrost, s katero potuje signal, in $\Delta\ t$ je razlika med časom prihoda med sprejemniki.

2.3.3 Moč prejetega signala

Moč prejetega signala (Received signal strength) je še eden izmed algoritmov za določanje lokacije. Uporablja moč prejetega signala iz naprav, ki se nahaja na znani lokaciji. Na podlagi, kako dober je sprejem signala iz naprav, lahko

določimo koliko je le ta oddaljena. Močnejši kot je signal, bližje smo napravi, ki oddaja, in na podlagi tega lahko določimo razdaljo. Ob tem potrebujemo za natančnost lokacije dodatne naprave, pri katerih lahko nato s trilateracijo pridobimo natančnejšo lokacijo.

Poglavje 3

Načini implementacije zavedanja lokacije v aplikacijah

Glede na veliko število tehnologij, ki so nam na voljo in smo jih opisali, se moramo pri implementaciji odločiti, kakšen način lokacijske zavednosti bomo imeli v aplikaciji. Na izbiro imamo kar nekaj možnosti, kot na primer geografsko ciljanje, lokacijska področja in svetilnike. Najpopularnejši implementaciji na mobilnih napravah sta s pomočjo svetilnikov, ki uporabljajo bluetooth tehnologijo in, lokacijska področja, ki uporabljajo kombinacijo tehnologij GPS, mobilnih omrežij in WiFi. V sledečem poglavju opišemo tri načine implementacij in njihovo delovanje ter analiziramo in primerjamo implementacije, ki so nam na voljo.

3.1 Geografsko ciljanje

Geografsko ciljanje pridobiva lokacijo uporabnika glede na IP naslov. Iz IP naslova lahko ugotovimo, iz katerega mesta ali regije prihaja določen uporabnik, in na podlagi tega dostavljamo potisna sporočila prilagojena tej lokaciji. Iz IP naslova lahko pridobimo lokacijo o uporabniku tako, da uporabimo informacije, ki so na voljo v oglasnih strežnikih. Ti vsebujejo velike podatkovne baze IP naslovov, ki so dodeljeni določenim ISP ponudnikom. Vsak

ISP ponudnik ima dodeljen interval IP naslovov, za katere lahko potem na podlagi teh podatkovnih baz določimo lokacijo, od kod dostopajo [44].

3.2 Svetilniki

Svetilnik (beacon) je mobilna naprava, ki se lahko namesti na želenih lokacijah. Na teh lokacijah nato naprava oddaja signal na določen interval, ki ga lahko mobilne naprave zaznajo [12]. Svetilniki se uporabljajo v sledečih primerih uporabe:

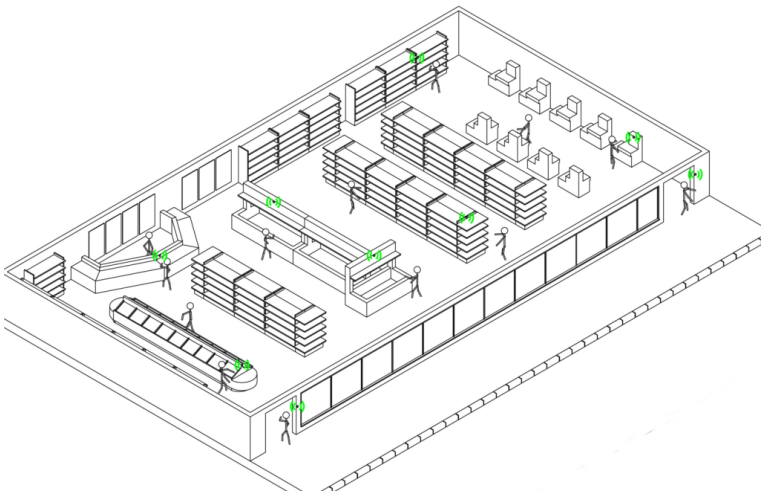
- navigacijo po zaprtem prostoru,
- sledenje predmetov,
- dostop brez ključa,
- trženje na mobilnih napravah,
- sledenje zaposlenim.

Glavni namen svetilnika je oddajanje signala na določen interval, da ga lahko mobilne naprave zaznajo. Svetilnik se ostalih naprav v svoji bližini ne zaveda in se ne poveže z njimi. V večini primerov svetilniki ne vsebujejo nobenih podatkov, prav tako se ne povezujejo z internetom. Informacijo, ki jo oddaja iBeacon podjetja Apple, je univerzalno enolični identifikator (UUID), major in minor [34]. V primeru Eddystone oddaja namespace, ki se uporablja kot UUID pri iBeacon, ter instance, ki predstavlja podatek kot major in minor skupaj pri iBeacon [30].
UUID je 16–bajten niz, s katerim lahko razlikujemo med prejetimi signali, od katere skupine svetilnikov prihajajo. Torej, vzemimo primer, kjer fiktivno podjetje Chocho d.o.o. želi v fiktivni trgovski verigi Spur d.o.o. tržiti svoje izdelke. Vsi svetilniki od podjetja Chocho d.o.o. bi vsebovali enak UUID, kar bi omogočilo mobilni napravi podjetja Chocho d.o.o. ugotoviti, katere vrste vsebino mora dostaviti glede na svetilnikov UUID.

Major vsebuje niz velikosti 2 bajta, ki nam pomaga ločiti podmnožico različnih svetilnikov v večji skupini. Torej v našem primeru, ki smo ga opisali, bi imelo podjetje Chocho d.o.o. v eni trgovini v trgovski verigi določen major, ki bi ga ločil od ostalih svetilnikov v preostalih trgovinah verige.
Minor vsebuje prav tako niz velikosti 2 bajta, ki določa vsak individualni svetilnik. Če se ponovno navežemo na naš primer, bi s pomočjo tega lahko ločili, kje se nahaja kakšen svetilnik. Mobilna naprava bi lahko s pomočjo minor ločila med svetilnikom, ki se nahaja pri mlečnih izdelkih, in svetilnikom, ki je pri sadju.

3.2.1 Delovanje mobilnih aplikacij s svetilniki

Sedaj, ko poznamo delovanje svetilnikov, bomo predstavili še delovanje aplikacij, ki jih uporabljajo. Tipičen primer uporabe svetilnikov je, da jih namestimo po prostoru, kot kaže slika 3.1. V našem primeru bomo uporabili trgovino, ki ima nameščene svetilnike po celotni trgovini, kjer oddajajo signal na določen interval. Ko potrošnik vstopi v trgovino z nameščeno aplikacijo na mobilnem telefonu, ta poskuša zaznati, če se v bližini nahaja kakšen svetilnik. Ko aplikacija zazna sporočilo svetilnika, sporoči identifikacijske parametre svojemu zalednemu sistemu. Ta sistem na podlagi identifikacije, v primeru iBeacon so to UUID, major in minor, ve, katera akcija se mora izvršiti. Akcije so lahko različna potisna sporočila, ki vsebujejo popuste glede na to, pri kateri polici se nahaja potrošnik. Prav tako lahko v tem primeru hranimo različne analitične podatke o potrošniku, kot na primer, kje se nahaja in kako potuje po trgovini. V našem primeru smo uporabili trgovino, lahko pa bi predstavili uporabo v letališčih, v muzejih, na koncertih in drugih podobnih lokacijah.



Slika 3.1: Primer postavitve svetilnikov v trgovini

3.3 Lokacijska področja

Lokacijsko področje deluje na način, da se določi virtualno območje okoli točke, ki jo določimo s pomočjo zemljepisne širine in dolžine. Lokacijsko področje se uporablja na način, da zazna vse uporabnike, ki so v bližini določene točke glede na radij, ki ga določimo. Torej moramo za delovanje poznati lokacijo uporabnika, kot tudi njegovo oddaljenost od točke interesa. Za zaznavanje lokacije uporabnika lahko uporabljamo tehnologije GPS, WiFi, beacons in mobilna omrežja (celične stolpe). S pomočjo programske opreme lahko glede na sprožilec izvedemo dejanje. Pri lokacijskem področju imamo na voljo tri sprožilce [35], ki so prikazani na sliki 3.2.

Vstopi – se sproži, ko uporabnik vstopi v virtualno območje.

Se zadržuje – se sproži, ko se uporabnik zadržuje določeno obdobje v območju. Obdobje lahko določimo v milisekundah.

Izstopi – se sproži, ko uporabnik zapusti virtualno območje.

Lokacijsko področje se uporablja, ko želimo vedeti, ali je uporabnik apli-

Slika 3.2: Primeri sprožilcev v lokacijskih področjih

kacije vstopil v našo trgovino ali kakšno drugo območje, ki smo ga določili. To storimo tako, da v aplikacijo, v katero smo implementirali monitoriranje, vstavimo lokacijsko področje [13]. Odvisno od tega, ali uporabnik aplikacije vstopi v območje, se zadržuje v njem ali ga zapusti, se sproži eno izmed dejanj, ki jih želimo. To je lahko potisno sporočilo, e-pošta ali kakšno drugo dejanje.

Koristi lokacijskih področij so, da lahko z njimi povečamo prodajo in izboljšamo zvestobo strank, saj lahko s kombinacijo s CRM sistemi vodimo marketinške akcije. Izvedemo lahko različne akcije v primerih, če stranka nekaj časa ni bila v naši trgovini. Takrat ustvarimo lokacijsko področje, ki se sproži, ko se stranka približa trgovini in ji ponudimo popust ob naslednjem nakupu.

Poznamo tri različne tipe lokacijskih področij [14]. Ti so lahko:

1. *Statični*, kjer je uporabnikova lokacija relativna glede na fiksno točko ali območje.

2. *Dinamčini*, kjer je uporabnikova lokacija relativna glede na podatkovni tok.

3. *Vsak z vsakim*, kjer je lokacija uporabnika relativna na druge uporabnike.

Statične se uporabljajo za različne marketinške kampanije, kjer uporabnika vabijo, da vstopi v trgovski center, ko je v bližini. Dinamične pridejo do izraza, kadar se lokacije monitoriranega območja spreminjajo, kot na primer parkirna mesta. Ta so lahko zasedena, v tem primeru območje ni monitorirano in obratno. Vsak z vsakim se danes uporabljajo v socialnih omrežjih, kjer nas lahko opozorijo, če se kakšen izmed prijateljev nahaja v naši bližini.

3.4 Analiza in primerjava različnih implementacij

Sedaj smo spoznali, kako delujejo geografsko ciljanje, lokacijska področja in svetilniki. Vse možnosti se danes uporabljajo v mobilnih aplikacijah, ampak ima vsaka svoje prednosti in slabosti, ki jih bomo predstavili in primerjali v tabeli 3.1, in primere uporabe v tabeli 3.2.

Za delovanje potrebujo mobilno aplikacijo na napravi. Lokacijsko področje uporablja za določanje lokacije GPS, WiFi ter mobilna omrežja, geografsko ciljanje uporablja IP naslov, medtem ko svetilniki delujejo s pomočjo bluetooth (BLE) tehnologije. Zaradi tega je poraba baterije na mobilnih napravah pri delovanju lokacijskih področij večja kot v primeru, kjer imamo aplikacijo s svetilniki ali geografskim ciljanjem. Aplikacije, kjer je implementirana lokacijska zavednost s pomočjo svetilnikov, morajo imeti zagotovljeno strojno opremo [40]. Danes so najpopularnejši svetilniki iBeacon[4] podjetja Apple in Eddystone[5], ki jih podpira Google. Za vsako lokacijo, ki jo želimo označiti, potrebujemo en svetilnik, katerih cena se giblje od 10 evrov naprej, medtem ko za postavitev lokacijskih točk s pomočjo lokacijskih področij ali geografskega ciljanja tega stroška nimamo.

Uporaba lokacijskih področij je priporočljiva za območja, ki so večja od 50 metrov, geografsko ciljanje za večja področja lokacij, kot so na primer določena mesta, medtem ko se svetilniki uporabljajo za manjše površine in zaprte prostore. Torej v primeru, ko želimo slediti, kolikokrat uporabnik obišče različne trgovine, ki so porazdeljene po državi, je bolje, da uporabimo

karakteristike	lokacijsko področje	geografsko ciljanje	svetilnik
Natančen na majhnih območjih	Ne	Ne	Da
Natančen na večjih območjih	Da	Da	Ne
Potrebuje aplikacijo za delovanje	Da	Da	Da
Potrebuje vključen Bluetooth	Ne	Ne	Da
Potrebuje GSM / WiFi / GPS signal	Da	Ne	Ne
Dodatni stroški	Ne	Ne	Da

Tabela 3.1: Tabela karakteristik [41]

lokacijska področja. Kadar pa želimo slediti gibanju uporabnika po svoji trgovini, se odločimo za implementacijo svetilnikov. Za uporabo geografskega ciljanja se odločimo, kadar želimo vedeti, kdaj se uporabnik nahaja v določenem kraju.

Za implementacijo na različnih platformah imamo pri implementaciji svetilnikov določene omejitve. Medtem, ko so lokacijska področja podprta tako na iOS, Android in Windows Phone, tega za svetilnike ne moremo trditi. Svetilnike podpira iOS od verzije 7.0 naprej, Android od verzije 4.4, medtem ko Windows Phone ne omogoča svetilnikov [39].

primer uporabe	priporočljiva izbira
Ko želimo vedeti, ali je uprabnik vstopil v konkurenčno trgovino.	Lokacijsko področje
Ko želimo vedeti, kje v trgovini se nahaja uporabnik.	Svetilniki
Ko želimo vedeti, ali je zaposleni zapustil delovno območje.	Lokacijsko področje
Ko želimo vedeti, v katerem kraju se nahaja uporabnik.	Geografsko ciljanje
Ko želimo uporabniku ponuditi popust, ko gre mimo izdelkov.	Svetilniki
Ko želimo avtomatiziratii prijavo na hotelski recepciji.	Svetilniki

Tabela 3.2: Tabela priporočljive implementacije za primer uporabe [41]

Poglavje 4

Implementacija lokacijskega področja

Na podlagi uporabe tehnologij, primerov uporabe in stroškov, ki nastanejo ob implementaciji, smo se odločili za lokacijsko zavednost s pomočjo lokacijskih področij. V nalogi smo implementirali lokacijska področja na platformo Android in iOS, kjer smo upoštevali dobre prakse za varčno rabo baterije [21] [11] in boljšo natančnost lokacijske zavednosti [22].

Aplikaciji, ki smo ju izdelali, smo načrtovali tako, da sta nam omogočili lažje testiranje delovanja in kasneje primerjave med različnima operacijskima sistemoma. Aplikaciji vsebujeta glavni zaslon, kjer se prikaže zemljevid, na katerem je razvidna naša trenutna lokacija. Prav tako so na zemljevidu prikazana vsa aktvina lokacijska področja in radij področja, kjer se sproži dogodek ob vstopu ali izstopu v lokacijsko področje. Aplikacija shranjuje podatke o spremembah trenutne lokacije uporabnika in časovni žig. S temi podatki smo nato lažje analizirali razliko med delovanjem. Spremlja tudi čas proženja dogodka v lokacijskem področju, ki ga imajo aplikacije za primerjavo odzivnosti na različnima operacijskima sistemoma, in delež sproženih lokacij, ki so bile podane.

Aplikaciji smo izdelali v orodju Android Studio za operacijski sistem Android in Xcode za operacijski sistem iOS. Prvo smo razvijali v programskem

jeziku Java, drugo pa v programskem jeziku Swift.

4.1 Implementacija Android

Za implementacijo lokacijske zavednosti v aplikacijo, ki deluje na Android platformi, uporabimo lokacijski API Google Location and Activity Recognition [19], ki je na voljo v Google Play storitvah. Aktualna različica API-ja v času pisanja je bila 11.2.0. Razvoj je potekal z Android Studio orodjem, ki je priporočljiv za izdelavo Android aplikacij. Google Play services nam omogoča, da uporabljamo vse zmogljivosti, ki so nam na voljo v sodobnih mobilnih napravah, ki bazirajo na operacijskem sistemu Android.

4.1.1 Android Studio

Android Studio je uraden IDE za razvoj Android aplikacij. Zasnovan je bil na osvnovi IntelliJ IDEA[6] orodja, ki ga je razvilo podjetje JetBrains. Android Studio je zasnovan tako, da nam olajša in pospeši razvoj Android aplikacij. Namestimo ga lahko na operacijskih sistemih Windows, Linux in macOS. Izdali so ga leta 2013 na konferenci Google I/O in njegova trenutna stabilna verzija je 2.3.3, ki je bila izdana junija 2017. IDE nam omogoča lažji razvoj, saj omogoča urejanje kode, razhroščevanje, testiranje in še veliko drugega.

4.1.2 Google Play services API

Na voljo imamo različne API-je, ki nam omogočajo uporabo funkcij, ki so na voljo v telefonu. Med pogosteje uporabljenimi funkcijami so Android Pay, Google Drive, Google Cloud Messaging, Google Maps in Google Location and Activity Recognition. V naši aplikaciji uporabljamo slednji. Google Play service APK vsebuje API-je, ki jih naša aplikacija uporablja in tečejo kot procesi v ozadju operacijskga sistema. Mi dostopamo do tega procesa preko odjemalca in proces izvrži akcije, ki jih od njega zahtevamo [20].

4.1.3 Razvoj lokacijske zavednosti na platformi Android

Pri razvoju smo najprej implementirali delovanje lokacijske zavednosti, kjer smo implementirali prikaz lokacije uporabnika in vseh lokacijskih področij na zemljevidu. Da smo lahko dodali zemljevid v aplikacijo, smo morali najprej dodati odvisnost na Google Play services API v *build.gradle* datoteko 4.1. V našem primeru smo potrebovali API *maps*, ki nam je omogočal prikaz zemljevida, in *locations*, ki je omogočil lokacijsko zavednost aplikacije.

Izvorna koda 4.1: Dodajanje odvisnosti na Google Play services API

```
compile 'com.google.android.gms:play-services-location:11.2.0'
compile 'com.google.android.gms:play-services-maps:11.2.0'
```

Da smo lahko prikazali zemljevid, smo pridobili Google Maps API ključ. Tega smo vstavili v *AndroidManifest.xml* datoteko, kjer je bil meta-data otrok application elementa vidnega v izseku programske kode 4.2. Postopek, da smo pridobili ključ, je bil sledeč:

1. Ustvarili smo projekt na Google API Console[7].
2. Izbrali projekt in kliknili Nadaljuj, da smo omogočili Google Maps Android API.
3. Na Credentials strani smo pridobili API ključ.
4. Ko smo pridobili ključ, smo vse shranili.

Če bi želeli, da ključ deluje samo na določeni Android aplikaciji, bi to lahko storili tako, da bi izbrali *Restrict key* med tretjim in četrtim korakom. V našem primeru se za omejitev nismo odločili.

Izvorna koda 4.2: Maps API ključ v AndroidManifest.xml

```
<meta-data
       android:name="com.google.android.geo.API_KEY"
       android:value="NAS_API_KLJUC"/>
```

Ko smo uspešno pridobili ključ, smo dodali *MapView* element v našo aplikacijo, kjer smo želeli prikazati zemljevid. Tega smo prikazali v *fragment-map.xml* layout datoteki 4.3.

Izvorna koda 4.3: Dodan MapView v layout datoteko

```
<com.google.android.gms.maps.MapView
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:id="@+id/map"
        />
```

Ko se nam je prikazal element, kjer je bil razviden zemljevid, je bil naslednji korak, da prikažemo lokacijo uporabnika in lokacijo vseh spremljanih področij. To nam je uspelo z Google Play service locations APIs, na katerega odvistnost smo dodali že v prejšnjih korakih v *build.gradle* datoteki. Aplikacije, ki koristijo informacije o lokaciji uporabnika, morajo pridobiti pravice za dostop do teh informacij. Te pravice nam lahko odobri uporabnik, od katerega jih moramo zahtevati. V *AndroidManifest.xml* datoteko smo vključili *uses-permission* element, kjer smo v našem primeru želeli *ACCESS FINE LOCATION* 4.4. Na voljo imamo dve različni pravici za dostop do lokacije. Pravica, ki smo jo izbrali nam omogoča natančnejšo pridobivanje lokacije uporabnika, kar je v primeru lokacijskih področij ključno. *ACCESS COARSE LOCATION* vrača podatke o lokaciji, ki vsebujejo slabše podatke o natančnosti, a privarčujemo pri porabi baterije, saj v tem primeru ne pridobiva lokacije samo s strani GPS [24].

Izvorna koda 4.4: Pravice do lokacijskih podatkov

```
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

V primeru, kadar naša aplikacija nima pravice, ki jo potrebuje, jo moramo zahtevati od uporabnika. V kodi smo implementirali *locationRequestPermission()* metodo, ki zahteva dovoljenje za dostop do lokacije uporabnika. V metodi najprej preverimo, ali imamo odobrene pravice. Kadar pravice niso

odobrene, zaprosimo uporabnika, da nam jih odobri, kar je razvidno iz izseka programske kode 4.5. Dobra praksa je, da to preverimo že ob zagonu aplikacije, saj v primeru, da nam pravic uporabnik ne odobri, lahko funkcionalnosti aplikacije, kjer so te pravice potrebne, onemogočimo [23].

Izvorna koda 4.5: Zahteva za lokacijske pravice

```
private void locationRequestPermission() {
    if (ContextCompat.checkSelfPermission(thisActivity,
                    Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(thisActivity,
                new String[]{Manifest.permission.READ_CONTACTS},
                MY_PERMISSIONS_REQUEST_READ_CONTACTS);
        }
    }
}
```

Ko smo implementirali pridobivanje pravic, smo lahko pričeli z implementacijo pridobivanja uporabnikove lokacije. Za to smo morali najprej ustvariti instanco *GoogleApiClienta* v naši *onCreate()* metodi. Instanco smo ustvarili s pomočjo *GoogleApiClient.Builder* razreda, ki omogoča uporabo specifičnih Google API-jev [16]. V našem primeru smo ustvarili metodo *getGoogleApiClient()*, ki inicializira *mGoogleApiClient* za lokacijsko storitev iz Google Play storitev 4.6. S pomočjo *mGoogleApiClient* smo lahko razvili pridobivanje lokacije uporabnika na določen interval. Za to smo uporabili razreda *LocationRequest* in *FusedLocationProviderApi*. To se zgodi, kadar se *mGoogleApiClient* uspe povezati in se pokliče metoda *createLocationRequest()* 4.7. Pri inicializaciji *LocationRequest* smo podali interval, na koliko časa v milisekundah bo aplikacija pridobivala posodobitve o lokaciji. Prav tako smo določili prioriteto zahtevkov, kar poda lokacijskemu API-ju namig o tem, kateri način pridobivanja lokacijskih podatkov naj uporabi. Mi smo izbrali *PRIORITY HIGH ACCURACY*, ki vrača najbolj natančen podatek o lokaciji uporabnika in v tem primeru v večini uporablja GPS za določanje lokacije [17]. Za

pridobivanje lokacije uporabnika smo nato implementirali *FusedLocationProviderApi* storitev, in tako ob klicu metode *requestLocation()*, kjer smo podali *mGoogleApiClient* in *mLocationRequest* kot parametra, pričeli slediti lokaciji uporabnika. Pridobljeni podatek smo shranjevali v spremenljivko *mCurrentLocation*, ki smo jo nato tudi uporabili za prikazovanje trenutne lokacije uporabnika na zemljevidu.

Izvorna koda 4.6: Ustvarjanje mGoogleApiClient instance

```
public void getGoogleApiClient() {
   mGoogleApiClient = new GoogleApiClient.Builder(this)
         .addApi(LocationServices.API)
         .addConnectionCallbacks(new
             GoogleApiClient.ConnectionCallbacks() {
            @Override
            public void onConnected(@Nullable Bundle bundle) {
               Log.wtf(TAG, "Connected to GoogleApiClient");
               createLocationRequest();
            }
            @Override
            public void onConnectionSuspended(int i) {
               Log.wtf(TAG, "Suspended connection to
                   GoogleApiClient");
            }
         })
         .addOnConnectionFailedListener(new
             GoogleApiClient.OnConnectionFailedListener() {
            @Override
            public void onConnectionFailed(@NonNull ConnectionResult
                connectionResult) {
               Log.wtf(TAG, "Failed to connect to GoogleApiClient" +
                     connectionResult.getErrorMessage());
            }
         })
         .build();
}
```

Izvorna koda 4.7: Pridobivanje lokacije uporabnika

```
protected void createLocationRequest() {
   LocationRequest mLocationRequest = LocationRequest.create()
         .setInterval(10000)
         .setFastestInterval(5000)
         .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
   mFusedLocationApi.requestLocationUpdates(mGoogleApiClient,
       mLocationRequest, new LocationListener() {
      @Override
      public void onLocationChanged(Location location) {
         mCurrentLocation = location;
      }
   });
}
```

Naslednji korak je bilo implementiranje funkcionalnosti lokacijskih področij. Kot smo že omenili, je za dobro delovanje potreben natančen podatek o lokaciji uporabnika, in zato uporabljamo *ACCESS FINE LOCATION* pravice. Za to je potrebno ustvariti Geofence objekt s pomočjo *Geofence.Builder()*, kjer določimo lokacijo, kje se nahaja lokacijsko področje s pomočjo zemljepisne širine in dolžine ter radij območja v metodi *setCiricularRegion()*, prikazani na izseku programske kode 4.8. Velikost radija je priporočljivo nastaviti na 100 metrov, zaradi različnih natančnosti določanja lokacije uporabnika in varčevanja s porabo baterije [18]. S pomočjo metode *.setExpirationDuration()* določimo, koliko časa želimo, da lokacijsko področje obstaja. Eden pomembnejših parametrov, ki jih moramo določiti za lokacijsko področje, je prehodni dogodek, zaradi katerega se področje sproži. Na voljo so *ENTER*, *EXIT* in *DWELL*. Dwell dogodek nam omogoča, da se dogodek za področje sproži, ko se uporabnik zadržuje v lokacijskem področju določeno časovno obdobje. Ko smo uspeli ustvariti seznam vseh lokacijskih področij, smo poklicali metodo *addGeofenceList()*, ki je ustvarila objekta GeofencingRequest in PendingIntent, ki sta potrebna kot parameter za *Locati-*

onServices.getGeofencingClient() 4.9, ki prične spremljati lokacije področij.

Izvorna koda 4.8: Ustvarjanje lokacijskih področij

```
public void createGeofenceList() {
   GeofenceList geofences = new GeofenceList();
   ArrayList<GeofenceModel> geofenceList = (ArrayList)
       geofences.returnGeofences();
   for (GeofenceModel geofence : geofenceList) {
      mGeofenceList.add(new Geofence.Builder()
            .setRequestId(geofence.getREQ_ID())
            .setCircularRegion(
                  geofence.getLocation().latitude,
                  geofence.getLocation().longitude,
                  geofence.getRadius()
            )
            .setExpirationDuration(geofence.getExpire())
            .setTransitionTypes(geofence.getTransition())
            .build());
   }
   addGeofenceList();
}
```

V metodi *addGeofenceList()* smo potrebovali *GeofencingRequest* objekt, ki mu podamo seznam lokacijskih področij, ki jih želimo spremljati. V njem smo določili *setInitialTrigger()*. Če se v trenutku, ko začnemo spremljati lokacijsko področje, nahajamo v enem izmed njih, nam bo to omogočilo, da se sproži dogodek področja 4.10. Prav tako smo v *addGeofenceRequest()* potrebovali *PendingIntent*, ki bo obravnaval dogodke, ki se bodo sprožili 4.12. Za delovanje tega smo izdelali razred *GeofenceService*, ki deduje od razreda *IntentService*. V tem razredu smo lahko v metodi *onHandleIntent()* pridobili podatke o lokacijskih področjih in dodali logiko, ki določa, kaj se zgodi ob vstopu, izstopu ali zadrževanju. Ko smo definirali *IntentService*, smo ga morali deklarirati v *AndroidManifest.xml* datoteki kot otrok application elementa razvidnega iz izseka programske kode 4.11.

Izvorna koda 4.9: Dodajanje lokacijskih področij v GeofencingClient servis

```
private void addGeofenceList() {
   mGeofencingClient.addGeofences(getGeofencingRequest(),
       getGeofencePendingIntent())
         .addOnSuccessListener(this, new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
               Log.wtf(TAG, "addGeofences() - succesfully added");
            }
         })
         .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
               Log.wtf(TAG, "addGeofences() - failed to add");
            }
         });
}
```

Izvorna koda 4.10: Ustvarimo GeofencingRequest objekt

```
private GeofencingRequest getGeofencingRequest() {
   GeofencingRequest.Builder builder = new
       GeofencingRequest.Builder()
         .setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER)
         .addGeofences(mGeofenceList);
   return builder.build();
}
```

Izvorna koda 4.11: Maps API ključ v AndroidManifest.xml

```
<service android:name="services.GeofenceService" />
```

Izvorna koda 4.12: Ustvarimo GeofencingRequest objekt

```
private PendingIntent getGeofencePendingIntent() {
   if (mGeofencePendingIntent != null) {
      return mGeofencePendingIntent;
   }
   Intent intent = new Intent(this, GeofenceService.class);
   return PendingIntent.getService(this, 0, intent,
       PendingIntent.FLAG_UPDATE_CURRENT);
}
```

Ko smo imeli vso logiko implementirano, smo lahko izdelali .apk datoteko s pomočjo Android Studio orodja in jo naložili na telefon, kjer smo pričeli s testiranjem lokacijskih področij. Za ustvarjanje .apk datoteke smo v Android Studio meniju *Build* izbrali možnost *Build APK*. APK je format datoteke, ki se uporablja za distribucijo in nameščanje mobilnih aplikacij na Android operacijski sistem.

4.2 Implementacija iOS

Lokacijsko zavednost v aplikaciji za iOS naprave smo implementirali s pomočjo *Core Location*[26] ogrodja. To ogrodje je vključeno v iOS SDK od verzije 2.0 naprej. Njegova glavna naloga je, da pridobiva geografsko lokacijo in orientacijo mobilne naprave. To počne s pomočjo senzorjev, ki so mu na voljo. Med razpoložljivimi so najpogosteje uporabljeni Wi-Fi, GPS in mobilna omrežja. Prav tako ima možnost implementacije lokacijske zavednosti s pomočjo svetilnikov, ki smo jih že omenili v začetku diplomskega dela.

4.2.1 Xcode

Xcode je IDE za razvoj aplikacij za operacijske sisteme podjetja Apple, med katere spadajo naprave, kot so Mac, iPhone, iPad, Apple Watch in še mnoge druge. Xcode podpira razvoj v različnih jezikih, kot so C++, Objective-

C, Java, Python in Swift. V Swift programskem jeziku smo razvili našo aplikacijo za iOS napravo. Prva verzija je izšla leta 2003, ki je bazirala na Project Builder[46] razvojnem okolju. Aplikacijo smo razvili na verziji 9, ki je izšla 19. septembra 2017. Xcode nam omogoča lahek razvoj aplikacij v jeziku Swift s pomočjo napotkov za bolj berljivo kodo. Prav tako je ena izmed večjih prednosti tega orodja, da Swift prevede kodo hitro in generira majhne binarne datoteke.

4.2.2 Razvoj lokacijske zavednosti na platformi iOS

Za pravilno delovanje aplikacije na platformi iOS smo najprej implementirali prikazovanje zemeljevida. Prikaz zemljevida nam omogoča MapKit ogrodje v Swift programskem jeziku. Na voljo je od verzije iOS 3.0 naprej.

Za prikaz moramo v naš projekt vključiti *MapKit* in *CoreLocation* ogrodji. V naši *ViewController.swift* 4.13 datoteki ustvarimo *MKMapView*[27], ki prikazuje zemljevid in vsebuje vmesnik za navigacijo po vsebini zemljevida. Ko imamo v aplikaciji viden zemljevid, implementiramo prikaz lokacije trenutnega uporabnika na zemljevidu s pomočjo *CLLocationManager*[25] objekta, ki ga vsebuje *CoreLocation* ogrodje, ki smo ga uvozili v *ViewController*.

Izvorna koda 4.13: ViewController datoteka

```
import UIKit
import MapKit

class ViewController: UIViewController {
    var window: UIWindow?
    var mapView: MKMapView?
    var locationManager: CLLocationManager?

    override func viewDidLoad() {
        super.viewDidLoad()

        self.window = UIWindow(frame: UIScreen.mainScreen().bounds)
        self.view.backgroundColor = UIColor.whiteColor()

        self.mapView = MKMapView(frame: CGRectMake(0, 20,
            (self.window?.frame.width)!, 300))
        self.view.addSubview(self.mapView!)

        self.locationManager = CLLocationManager()
        if let locationManager = self.locationManager {
            locationManager.delegate = self
            locationManager.desiredAccuracy =
                CLLocationAccuracyBestForNavigation
            locationManager.requestAlwaysAuthorization
            locationManager.distanceFilter = 5
            locationManager.startUpdatingLocation()
        }
    }
}
```

Za vidnost uporabnikove lokacije s pomočjo *CLLocationManager* 4.14 moramo nastaviti posebna dovoljenja v *Info.plist* datoteki. Za naš primer do-

damo ključ *NSLocationAlwaysUsageDescription* in v vrednost vnesemo besedilo, ki ga želimo prikazati, ko se pojavno okno za dovoljenja izpiše na mobilni napravi.

Izvorna koda 4.14: Prikaz lokacije uporabnika

```
func locationManager(manager: CLLocationManager,
    didUpdateToLocation newLocation: CLLocation, fromLocation
    oldLocation: CLLocation) {
    if let mapView = self.mapView {
        let region =
            MKCoordinateRegionMakeWithDistance(newLocation.coordinate,
            200, 200)
        mapView.setRegion(region, animated: true)
        mapView.showUserLocation = true
    }
}
```

Za zaključek smo v aplikaciji implementirati še funkcionalnost lokacijskih področij s pomočjo *CLCiricularRegion* objekta, ki je del *CoreLocation* ogrodja. Za ta objekt definiramo lokacijo področja, njegov radij in identifikator, ki sistemu omogoča razlikovanje med različnimi lokacijskimi področji. Za vsak *CLCiricularRegion* lahko določimo, ali želimo, da se sproži ob vstopu ali izstopu naprave v lokacijsko področje. To nam omogočata boolean vrednosti v razredu, *notifyOnEntry* in *notifyOnExit*.

Izvorna koda 4.15: Ustvarjanje lokacijskega področja

```
func createGeofence(center: CLLocationCoordinate2D, identifier:
    String, radius: CLLocationDistance) {
    let maxDistance =
        locationManager.maximumRegionMonitoringDistance
    let geofence = CLCircularRegion(center: center,
                                            radius: radius,
                                            identifier:
```

```
                                                        identifier)
        geofence.notifyOnEntry = true
        geofence.notifyOnExit = false

        return geofence
    }
```

V metodi *createGeofence* ustvarimo različna lokacijska področja. Za ustvarjeno lokacijsko področje je nato potrebno omogočiti spremljanje, kar mi storimo v metodi *startMonitoring*, ki sprejme objekt s podatki o področju za spremljanje. Naš objekt *Geofence* vsebuje podatek o radiju, koordinate in identifikator za želeno področje, ki ga želimo spremljati.

Izvorna koda 4.16: Pričetek sledenja lokacijskega področja

```
    func startMonitoring(geofence: Geofence) {
        if !CLLocationManager.isMonitoringAvailable(for:
            CLCircularRegion.self) {
            showAlert(withTitle:"Napaka", message: "Lokacijsko
                podrocje ni podprto!")
            return
        }

        if CLLocationManager.authorizationStatus() !=
            .authorizedAlways {
            let message = """
                Da bi program deloval morate omogociti dostop
                do podatkov o lokaciji naprave.
                """
            showAlert(withTitle:"Opozorilo", message: message)
        }
        let geofenceRegion = createGeofence(center:
            geofence.coordinate, identifier:
            geofence.identifier,radius: geofence.radius)
```

```
        locationManager.startMonitoring(for: geofenceRegion)
    }
```

V metodi najprej preverimo, ali naprava podpira lokacijska področja. Sledi preverjanje, ali je uporabnik aplikaciji dodelil potrebne pravice, da lahko dostopa do podatkov o lokaciji. V primeru, da nam pravice niso dodeljene, lahko lokacijska področja še vedno dodajamo, ampak zaznavali jih bomo šele takrat, ko bo uporabnik dovolil dostop do lokacije naprave. S pomočjo metode *createGeofence* ustvarimo objekt *CLCircularRegion*, ki ga prijavimo v *CLLocationManager* [15].

Za prenehanje sledenja lokacijskemu področju smo ustvarili metodo *stopMonitoring*, ki sprejme identifikator kot argument. S pomočjo tega argumenta poiščemo želeno lokacijsko področje v našem *CLLocationManage*r objektu in prenehamo s sledenjem.

Izvorna koda 4.17: Prenehanje sledenja lokacijskega področja

```
    func stopMonitoring(identifier: String) {
        for region in locationManager.monitoredRegions {
            guard let geofenceRegion = region as? CLCircularRegion,
                geofenceRegion.identifier == identifier else {
                    continue }
            locationManager.stopMonitoring(for: circularRegion)
        }
    }
```

4.3 Razlike med operacijskima sistemoma

Pri implementaciji smo opazili dve razliki, ki močno vplivata na načrtovanje aplikacije in primere uporabe. V primeru dodajanja različnih lokacijskih področij smo na napravah z operacijskim sistemom Android omejeni na 100 lokacij, medtem ko v primeru iOS operacijski sistem omogoča 20 lokacij. To moramo pri načrtovanju aplikacije upoštevati, saj se lahko zgodi, da bo v

našem primeru uporabe potrebnih več kot 20 lokacij. Rešitev za takšen primer smo našli v tem, da imamo zaledni sistem, ki se mu pošlje zahtevek s trenutno lokacijo naprave. Na podlagi tega nam zaledni sistem vrne 20 najbližjih točk trenutni lokaciji naprave, kadar smo na iOS operacijskem sistemu. V primeru, da imamo aplikacijo nameščeno na Android operacijski sistem, nam lahko vrne 100 lokacij. Zaradi tega moramo imeti čas posodabljanja lokacij na aplikaciji, ki je nameščena na iOS operacijskem sistemu krajši kot na Android. Prav tako so potem pogostejše poizvedbe na zaledni sistem in posledično večja poraba podatkov ter baterije.

Drugi primer, kjer smo opazili razliko, ki vpliva na delovanje, je, da Android omogoča proženje dogodkov, ko se uporabnik zadržuje določen čas v lokacijskem področju. Naprave, kjer je operacijski sistem iOS, tega delovanja ne omogočajo. Primerna rešitev za ta problem se nam je zdela, da dodamo v aplikacijo še števec, ki meri čas od trenutka, ko uporabnik vstopi v področje in po pretečenem času sproži dogodek, ki je predviden za to lokacijsko področje.

Obe implementaciji smo testirali s testnima aplikacijama na mobilnih napravah Huawei P8 Lite in Apple iPhone 7. Analizo delovanja predstavimo v poglavju 5.

Poglavje 5

Analiza delovanja na različnih operacijskih sistemih

5.1 Zasnova eksperimenta

Eksperiment smo izvedli na dveh mobilnih napravah z različnima platformama, kjer smo uporabili tri različne načine transporta. Testiranje je potekalo v različnih okoljih in različnih velikostih lokacijskih področij.

Odzivnost in natančnost delovanja na platformah Android in iOS smo testirali v urbanem in podeželskem okolju. Za urbano okolje razvidno iz slike 5.1 smo izbrali tri lokacije na območju Šiške v Ljubljani. Medtem ko smo teste za podeželje razvidno na sliki 5.2 izvajali v vasi Pameče, ki se nahaja v koroški regiji. V vsakem okolju smo testirali delovanje s pomočjo treh različnih načinov transporta. Z avtomobilom pri hitrosti 70 km/h, s kolesom pri hitrosti 20 km/h in pri hitri hoji s hitrostjo 8 km/h. Skozi celoten eksperiment smo si s pomočjo merilnikov hitrosti prizadevali vzdrževati konstantno hitrost. Ustvarili smo tri različna lokacijska področja različnih premerov. Največji je bil velikosti 100 m, sledila sta mu še srednji z velikostjo 50 m in najmanjši z velikostjo 20 m. Pot v podeželskem okolju smo prepotovali v popoldanskem času, medtem ko smo pot v urbanem okolju prepotovali zjutraj. Potovali smo ob jasnem vremenu za najboljše delovanje GPS sistema [31].

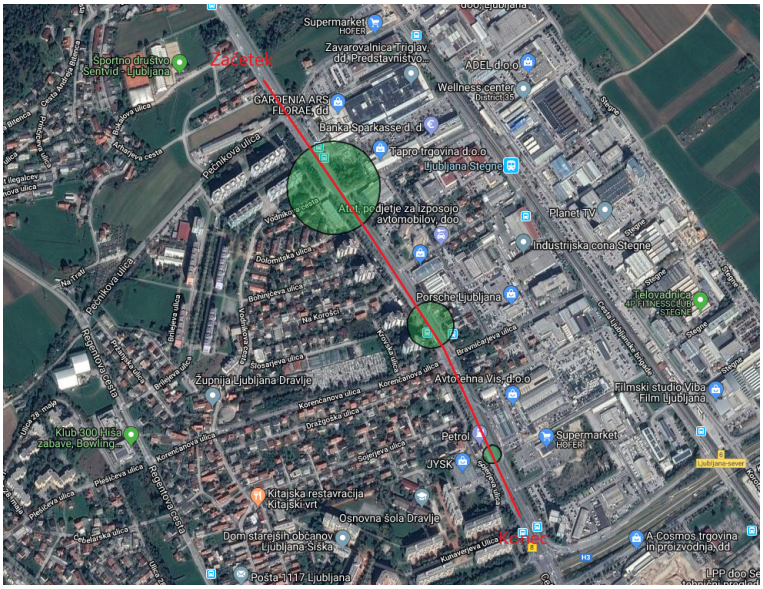
Naprava, ki uporablja platformo Android, je bila Huawei P8 Lite z različico sistema Android 6.0 Marshmallow [9]. S platformo iOS smo uporabljali napravo Apple iPhone 7, ki ima različico operacijskega sistema iOS 12.1 [10].

Zaznavo vstopa v lokacijsko področje smo za lažjo analizo v aplikaciji zapisovali v posebno datoteko *geofences.log*, ki je prikazana na izseku 5.1.

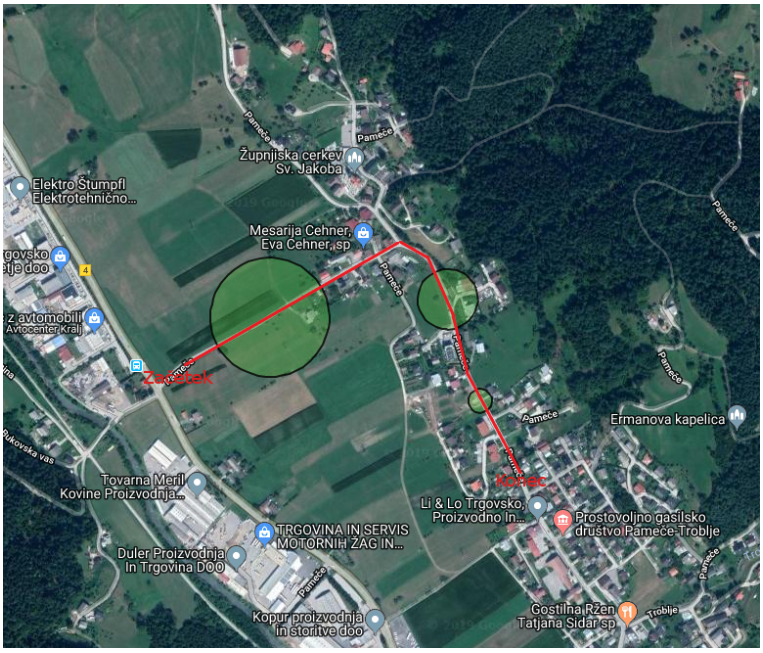
Izvorna koda 5.1: Izsek zapisa v datoteko za zaznavo lokacijskih področij na Android napravi s kolesom

```
2019-03-08 7:14:04,553 You have entered the geofence.
2019-03-08 7:18:42,174 You have entered the geofence.
2019-03-08 7:20:56,739 You have entered the geofence.
```

Hitrosti, ki so bile predvidene za testiranje delovanja, smo merili s pomočjo merilnikov hitrosti gibanja. V avtu smo uporabljali merilnik hitrosti, ki je prikazan na sprednjem zaslonu, medtem ko smo za merjenje hitrosti na kolesu in pri hitri hoji uporabljali aplikacijo Strava [8], ki se uporablja za merjenje hitrosti na mobilni napravi in za izračun hitrosti uporablja GPS. Skozi testno fazo smo si s pomočjo merilnikov prizadevali, da smo ohranjali konstantno hitrost, kjer dopuščamo možnost, da je prišlo do manjših odstopanj pri hitrosti.



Slika 5.1: Postavitev lokacijskih področij in poti v urbanem okolju



Slika 5.2: Postavitev lokacijskih področij in poti v podeželskem okolju

5.2 Rezultati

V tabeli 5.1 in 5.2 smo predstavili rezultate našega eksperimenta.

	70 km/h	20 km/h	8 km/h
100 m	2/2	2/2	2/2
50 m	1/2	2/2	2/2
20 m	0/2	1/2	1/2

Tabela 5.1: Tabela števila proženja lokacijskega področja na Android napravah

V primeru, kadar imamo lokacijsko področje z večjim premerom, je zaznava na obeh platformah dobra. Težava nastopi v kombinaciji hitrosti z manjšo velikostjo področja. Takrat se zaznava zmanjša, saj v našem primeru pri hitrosti 70 km/h in radiju 20 m vstopa v področje ni zaznala aplikacija niti na Android niti na iOS platformi. Pri hitrosti 20 km/h je od dveh ponovitev zaznala en vstop v urbanem okolju aplikacija na Android in iOS platformi. Mobilna naprava z iOS operacijskih sistemom je zaznala pri hitrosti 8 km/h oba vstopa, Android aplikacija pa samo tistega, ki se je zgodil v urbanem okolju. Kadar smo imeli radij aplikacije večji, je zaznalo vsak vstop v obeh okoljih.

	70 km/h	20 km/h	8 km/h
100 m	2/2	2/2	2/2
50 m	2/2	2/2	2/2
20 m	0/2	1/2	2/2

Tabela 5.2: Tabela števila proženja lokacijskega področja na iOS napravah

5.3 Diskusija

Na podlagi rezultatov smo prišli do zaključka, da aplikacija dobro zaznava lokacijsko področje, kadar je radij večji in hitrost gibanja manjša. Med napravama smo ugotovili, da je mobilna naprava z Android operacijskim sistemom zaznala 13 od 18 lokacijskih področij, medtem ko je naprava z operacijskim sistemom iOS zaznala 15 od 18 lokacijskih področij. Iz tega lahko zaznamo, da je v našem primeru naprava z iOS operacijskim sistemom delovala bolje, ampak da do očitnih razlik v zaznavi med napravama ne prihaja.

Pri primerjavi implementacij smo spoznali, da se za večja področja in zunanje okolje raje uporabljajo lokacijska področja, ko pa želimo natančne lokacije znotraj manjših območjih, za to uporabljamo svetilnike. Velika slabost uporabe svetilnikov v primerjavi z uporabo lokacijskih področij je večji strošek implementacije, saj je za delovanje potrebno svetilnike kupiti.

Velikokrat se lahko ob določanju lokacije mobilne naprave pripetijo napake tehnologije GPS. Med pogostejše spadajo odboj signala, ki se dogaja v urbanih okoljih, saj se signal od betonskih zgradb odbija. Prav tako se lahko zgodi napaka, kadar ima mobilna naprava napačno določeno uro, kar posledično privede do napačnega izračuna razdalje. In še ena izmed napak, ki se lahko pripeti, je atmosferska upočasnitev signala, ki nam posledično ne zagotavlja natančne informacije o hitrosti signala. Te napake lahko povzročijo napačno določeno lokacijo mobilne naprave.

V našem eksperimentu nismo zasledili napak tehnologije GPS, saj je zaznava aplikacije delovala bolje v urbanem okolju kot na podeželju, kjer je manj betonskih zgradb. Ob eksperimentu smo bili pazljivi, da imajo naprave pravilno določene ure, s čimer smo zagotovili, da so bili izračuni razdalje pravilni.

Poglavje 6

Zaključek

Cilj te diplomske naloge je bil raziskati različne tehnologije, ki nam omogočajo pridobivanje lokacije na mobilnih napravah, in raziskati različne načine implementacij lokacijske zavednosti v mobilnih aplikacijah. Cilj smo realizirali tako, da smo implementirali aplikaciji z lokacijsko zavednost za platformi Android in iOS. Analizirali smo različna pristopa in izbrali glede na naše potrebe najustreznejšega. Delovanje smo uspešno testirali v realnem scenariju in na podlagi tega uspeli predstaviti rezultate delovanja lokacijskih področij na obeh platformah.

V diplomski nalogi smo najprej spoznali različne načine lokacijske zavednosti na mobilnih napravah. Podrobneje smo spoznali štiri načine pridobivanja podatkov o lokaciji in delovanje trilateracije. Ob tem smo primerjali, kako natančne so različne tehnologije pridobivanja podatkov o lokaciji. Sledila je primerjava dveh različnih možnosti implementacije lokacijskih aplikacij in njune prednosti ter slabosti. Zaradi naših potreb smo se odločili, da bomo implementirali lokacijsko zavednost v naši aplikaciji s pomočjo lokacijskih področjih, kjer smo v nadaljevanju predstavili možnost implementacije na platformi Android in iOS. Ob implementaciji in razvoju smo opazili razlike med operacijskima sistemoma, ki smo jih tudi podrobneje predstavili. Za sklepno dejanje v diplomski nalogi smo testirali delovanje na obeh operacijskih sistemih v različnih okoljih in pod različnimi pogoji.

V bodoče bomo poskusili še z implementacijo delovanja lokacijske zavednosti v aplikaciji s pomočjo svetilnikov in izboljšave trenutnih zmogljivosti lokacijskih področij. Večino tega bomo poskusili implementirati ob izboljšavah s prihodom novih funkcionalnosti na platformah Android in iOS. Med izboljšave spadajo natančnejše določanje lokacije uporabnika in izboljšanje delovanja aplikacije na področju varčne rabe baterije.

Literatura

[1] Dosegljivo: http://www.google.com/. [Dostopano 6. 3. 2019].

[2] Dosegljivo: http://www.skyhookwireless.com/. [Dostopano 6. 3. 2019].

[3] Dosegljivo: https://www.apple.com/. [Dostopano 6. 3. 2019].

[4] Dosegljivo: https://en.wikipedia.org/wiki/IBeacon. [Dostopano 6. 3. 2019].

[5] Dosegljivo: https://en.wikipedia.org/wiki/Eddystone. [Dostopano 6. 3. 2019].

[6] Dosegljivo: https://www.jetbrains.com/idea/. [Dostopano 6. 3. 2019].

[7] Dosegljivo: https://console.developers.google.com/. [Dostopano 6. 3. 2019].

[8] Dosegljivo: https://www.strava.com/. [Dostopano 7. 3. 2019].

[9] Android Marshmallow 6.0. Dosegljivo: https://www.android.com/versions/marshmallow-6-0/. [Dostopano 6. 3. 2019].

[10] Apple iOS 7. Dosegljivo: https://www.apple.com/ios/ios-12/. [Dostopano 6. 3. 2019].

[11] Mohammed Alsaqer, Brian Hilton, Tom Horan, and Omar Aboulola. Performance assessment of geo-triggering in small geo-fences: Accuracy,

reliability, and battery drain in different tracking profiles and trigger directions. *Procedia Engineering*, 107:337 – 348, 2015. Humanitarian Technology: Science, Systems and Global Impact 2015, HumTech2015.

[12] Seung Hyo Baek and Seung Hyun Cha. The trilateration-based ble beacon system for analyzing user-identified space usage of new ways of working offices. *Building and Environment*, 149:264 – 274, 2019.

[13] Barry Berman. Planning and implementing effective mobile marketing programs. *Business Horizons*, 59(4):431 – 439, 2016.

[14] Aleksander Buczkowski. Location-Based Services – Technologies. Dosegljivo: `http://geoawesomeness.com/knowledge-base/location-based-services/location-based-services-technologies/`. [Dostopano 10. 8. 2017].

[15] Apple Developer. Region Monitoring and iBeacon. Dosegljivo: `https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/LocationAwarenessPG/RegionMonitoring/RegionMonitoring.html`. [Dostopano 12. 1. 2018].

[16] Android Developers. Accessing Google APIs. Dosegljivo: `https://developers.google.com/android/guides/api-client#start_an_automatically_managed_connection`. [Dostopano 19. 8. 2017].

[17] Android Developers. Changing Location Settings. Dosegljivo: `https://developer.android.com/training/location/change-location-settings.html`. [Dostopano 19. 8. 2017].

[18] Android Developers. Creating and Monitoring Geofences. Dosegljivo: `https://developer.android.com/training/location/geofencing.html`. [Dostopano 20. 8. 2017].

[19] Android Developers. Google Location and Activity Recognitions. Dosegljivo: https://developers.google.com/android/reference/com/google/android/gms/location/package-summary. [Dostopano 12. 9. 2017].

[20] Android Developers. Google Play services. Dosegljivo: https://developers.google.com/android/guides/overview. [Dostopano 17. 9. 2017].

[21] Android Developers. Optimize for battery life. Dosegljivo: https://developer.android.com/topic/performance/power. [Dostopano 22. 2. 2019].

[22] Android Developers. Optimize location for battery. Dosegljivo: https://developer.android.com/guide/topics/location/battery. [Dostopano 22. 2. 2019].

[23] Android Developers. Requesting Permissions at Run Time. Dosegljivo: https://developer.android.com/training/permissions/requesting.html. [Dostopano 18. 8. 2017].

[24] Android Developers. Specify App Permissions. Dosegljivo: https://developer.android.com/training/location/retrieve-current.html#permissions. [Dostopano 18. 8. 2017].

[25] Apple Developers. CLLocation Manager Documentation. Dosegljivo: https://developer.apple.com/documentation/corelocation/cllocationmanager. [Dostopano 6. 3. 2019].

[26] Apple Developers. Core Location Documentation. Dosegljivo: https://developer.apple.com/documentation/corelocation. [Dostopano 6. 3. 2019].

[27] Apple Developers. MapKit Map View Documentation. Dosegljivo: https://developer.apple.com/documentation/mapkit/mkmapview. [Dostopano 6. 3. 2019].

[28] Richard D. Easton. *GPS Declassified : From Smart Bombs to Smartphones.* Potomac Books, 2013.

[29] Vincent Gao. Proximity And RSSI. Dosegljivo: `https://blog.bluetooth.com/proximity-and-rssi`. [Dostopano 10. 8. 2017].

[30] Agnieszka Gasiorek. Beacon Strategy Guide – UUID, Major, Minor. Dosegljivo: `https://kontakt.io/blog/beacon-id-strategy-guide-quick-deployment/`. [Dostopano 12. 8. 2017].

[31] Thierry Gregorius and Geoffrey Blewitt. The effect of weather fronts on gps measurements. *GPS world*, 9(5):52–60, 1998.

[32] Techopedia Inc. Location-Aware Application. Dosegljivo: `https://www.techopedia.com/definition/30836/location-aware-application`. [Dostopano 20. 7. 2017].

[33] Techopedia Inc. Location Awareness. Dosegljivo: `https://www.techopedia.com/definition/16328/location-awareness`. [Dostopano 20. 7. 2017].

[34] Patrick Leddy. 10 Things About Bluetooth Beacons You Need To Know. Dosegljivo: `http://academy.pulsatehq.com/bluetooth-beacons`. [Dostopano 12. 8. 2017].

[35] Patrick Leddy. 7 Things About Geofencing You'll Kick Yourself For Not Knowing. Dosegljivo: `http://academy.pulsatehq.com/7-things-about-geofencing`. [Dostopano 13. 8. 2017].

[36] Luo Li and Deng Ping. Study on wifi indoor location techniques based on android. *[D] Southwest Jiaotong University*, 5, 2014.

[37] Phil Locke. Cell Tower Triangulation – How it Works. Dosegljivo: `https://wrongfulconvictionsblog.org/2012/06/01/cell-tower-triangulation-how-it-works/`. [Dostopano 10. 8. 2017].

[38] BeaconZone Ltd. Determining Location Using Bluetooth Beacons. Dosegljivo: https://www.beaconzone.co.uk/bluetoothleposition. [Dostopano 10. 8. 2017].

[39] Oleg Morajko. Beacons vs Geofencing, Which Way to Go? Dosegljivo: https://www.proximity.directory/blog/beacons-vs-geofencing-which-way-to-go. [Dostopano 14. 8. 2017].

[40] Oleg Morajko. Geofencing Vs Beacons: What's The Difference? Dosegljivo: http://blog.apps-builder.com/geofencing-vs-beacons-whats-the-difference/. [Dostopano 14. 8. 2017].

[41] Oleg Morajko. How to use geofences and beacons to engage mobile users. Dosegljivo: https://www.mocaplatform.com/blog/how-to-use-geofences-and-beacons-to-engage-mobile-users. [Dostopano 22. 2. 2019].

[42] Brian O'Keefe. Finding Location with Time of Arrival and Time Difference of Arrival Techniques. Dosegljivo: https://sites.tufts.edu/eeseniordesignhandbook/files/2017/05/FireBrick_OKeefe_F1.pdf. [Dostopano 27. 2. 2019].

[43] Anand Ranganathan, Jalal Al-Muhtadi, Shiva Chetan, Roy Campbell, and M. Dennis Mickunas. Middlewhere: A middleware for location awareness in ubiquitous computing applications. *Middleware 2004*, pages 397–416, 2004.

[44] GeoEdge University. How does Geo-targeting Work? Dosegljivo: https://www.geoedge.com/meetus_university/40/how-does-geo-targeting-work. [Dostopano 22. 2. 2019].

[45] Frank Stephen Tromp Van Diggelen. *A-GPS : Assisted GPS, GNSS, and SBAS.* Artech House GNSS Technology and Applications Library. Artech House, Inc, 2009.

[46] Wikipedia. Project Builder. Dosegljivo: https://en.wikipedia.org/wiki/Project_Builder. [Dostopano 6. 3. 2019].

[47] Tracy V. Wilson. How GPS Phones Work. Dosegljivo: http://electronics.howstuffworks.com/gps-phone.htm. [Dostopano 8. 8. 2017].

[48] Shixiong Xia, Yi Liu, Guan Yuan, Mingjun Zhu, and Zhaohui Wang. Indoor fingerprint positioning based on wi-fi: An overview. *ISPRS Int. J. Geo-Information*, 6:135, 2017.

[49] Fred Zahradnik. An Explanation of Wi-Fi Triangulation. Dosegljivo: https://www.lifewire.com/wifi-positioning-system-1683343. [Dostopano 25. 7. 2017].